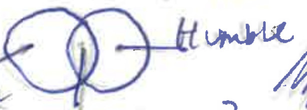
(5/5) 8:35 am (Shower ~~to be est~~ 30 min) /// The Value of Time Ⓛ & Expression ü ºº;) /// "yare Good!" Thanks, I try, but the effort; towards such perfection is a lofty goal, is not ;) // Combining/re-alizing the 3+ timeline combination/synthesis/integ-ration (Ultimate Matrix machine (stark bilocation/sleeper training/alt time line) Doom + et al) reality, Galactus, ultron & /gears base // <-> Mechanical clockwork reality may be uncomfort-able, but Galactus "decides" to leave - like "The atomic man (blue Krsna asche-type from watchmen /// Goal uniting all time-lines // although his Doom evil B/witch mother sold her soul, sudden Doom claims righteous self-sacrifice+ and teaches of the ever growing releasing soul/shit (distinction between soul, spirit, body △) / 905 so off Bandy ;)

(5/5) still happy for ANY est. short line, exciting day → Time wait line/time is STILL VALUABLE even to Joe Blow the LOWLIEST OF LOW (Q: is Humor still HUMility ?!)

// — Synthesis?



/// Update ~9:10

vameer/Snake bite says maybe 10 min more /// GRATEFUL /// HOW TO ASSIST/BREAK/RE/REVERSE/ENGINEER NON BELIEVERS/non KNOWERS ?* if they dont believe I'm Real or dont KNOW* I'm real THEN become the UNREAL /UNKNOWN ∞ 0 $/ & I'm ~~like~~ a Turbo Engine

☐ Research
☒ Stan lee's "final" novels ;)
+ to sin ☐ V for Vengeance* Vendetta family

☒ Recovery cafe open @ 12
Library (Dub

☒ Research day
☒ male ♂ ♀ Geese
☒ find my Tobacco contain new
☒ ~~front~~? day - 5pm
☒ 10:55 - laundry ~10 min
☒ 10-1105
☒ current 1007
☒ 1048 wash to dry
☒ research Dr. manhattan end stories

☐ Wine
☒ GATTACA* Genetic Discrim.
☐ SAGE ME
[G-Force Theory If I DONT GO FAST I GO TO SLEEP?

(5/4) monday ☐ the hangover ☐ asy ☒ observer effect studies AFTER (first observed by A.I. then after, how much different is it? if the observer uses an A.I. to watch all & only uses critical eye bias after event is recorded) compared to same event/action/behavior observed DURING action ☒ Senaca Tribe &/or Orator

Appendix B. Program listings

The program gauss displays

- the day of the week ({Mon, Tue, Wed, Thu, Fri, Sat, Sun})
- the Gregorian date,
- the Julian date (JC),
- the day of the year (D#),
- the week of the year (W#),
- the Julian day number (J#),
- the Unix day (X#)

The input syntax is `date [offset]`, where *date* is in ISO 8601 format `year-mm-dd` and *offset* can be any integer. Weeks run from Monday to Sunday, its numbering follows ISO.

When experimenting with some input you will see that gauss is tolerant to some false dates (correcting them): asking for `1999-2-29` it correctly answers `1999-03-01`. Empty input will terminate the program.

B.1. **Earlier implementations.** The subroutines go back to an implementation in REXX in the late 1980s on the IBM VM/CMS operating system. They were used in a program to generate weekly statistics of defects in a project. Then, at the wake of another leap year (1996), I put them all together into a small REXX program (`gauss.rexx`, see below in section B.3 for a recent version) and wrote up a *rationale* of the algorithm and a proof of the formula. This rough explanation of *Gauss'* formula constituted the original content at my Web site (June 1999).

In March 2004 I ported the program to Perl, and in March 2009 to C. Since 2004 the *Julian day number* and the Julian Calendar date was included.

The subroutine `date` implements the operation of the integers on the dates and `offset` returns the day of the year of a date. The subroutine `leap` implements the Gregorian leap year rules and `weekday` returns the result of *Gauss'* formula. I deviate from the text above and use as numbering of the weekdays Monday = 0, ..., Sunday = 6. This way the *Julian day number* is ≡ to the weekday. The subroutine `julian` calculates the Julian day number.

For positive y we have in REXX `c = y % 100` (integer divide), and the remainder is `g = y // 100`. This is false for negative y, when REXX yields `y//100 = -[-y/100]` instead. C suffers from the same wrong floor function as the other programming languages: the Euclidean algorithm gives the formula $a = q \cdot b + r$ with $q = \lfloor a/b \rfloor$ and $0 \le r < b$ for positive b. If a is negative the C function a/b gives the wrong result except if b divides a. The routines `quot` and `mod` correct the handling of the floor function for negative arguments (and I allow for negative years).

These implementations have been run on various platforms:

- REXX on VM, OS/2, DOS, Windows, Linux
- Perl on Linux, AiX, Windows
- C on Windows and Linux.

B.2. **Program listing `gauss.c`.**

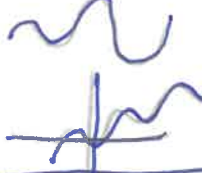
```
/* ------------------------------------------------------------------ *

   Module:     gauss.c
```

```
   Description:

         This program explores Gauss' formula for weekday calculation
         in the proleptic Gregorian Calendar.

         For a detailed explanation see my calendar article:
         https://berndt-schwerdtfeger.de/wp-content/uploads/pdf/cal.pdf

         Input:  year-mm-dd [offset] (ISO 8601 Format)

         Output: Weekday, date, Julian Calendar date, day of the year,
                 week of the year, Julian day number, Unix day number

   Subroutines: date, offset, weekday, julian, leap, quot, mod

   Sample: gauss 2010-01-0 +120
         Fri 2010-04-30, JC 2010-04-17, D# 120 W# 17 J# 2455317 X# 14729

  ------------------------------------------------------------------

   Copyright (C) 2010 Berndt E. Schwerdtfeger

   Licensed under the Apache License, Version 2.0 (the "License");
   you may not use this file except in compliance with the License.
   You may obtain a copy of the License at

         https://www.apache.org/licenses/LICENSE-2.0

   Unless required by applicable law or agreed to in writing, software
   distributed under the License is distributed on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   See the License for the specific language governing permissions and
   limitations under the License.

  ------------------------------------------------------------------ */

#include <stdio.h>
#include <stdlib.h>

// set constants

char *wd[] = {"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"};
int em[13] = {0,0,31,59,90,120,151,181,212,243,273,304,334};
const int JULIAN = 0;
const int GREGOR = 1;

// prototype statements for functions

void date(int,long,long*,int*,int*); // type, offset, year, month, day
long offset(long,int,int);           // = offset into year y
int  weekday(long,int,int);          // = {0|1|2|3|4|5|6}, 0 = Monday
long julian(long,int,int);           // = number of days since JC -4712/01/01
```

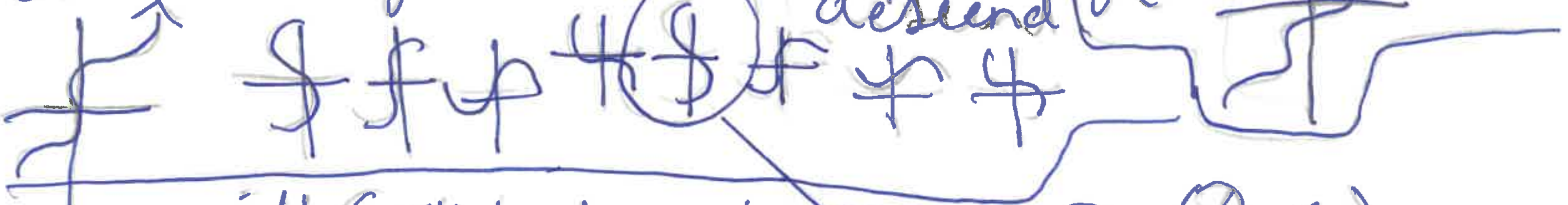
```
int  leap(int,long);                 // = 1 if year is a leap year
long quot(long,long);                // = [a/b] (corrected: works if a < 0)
long mod(long,long);                 // = a - quot(a,b)*b

// ------------------------------------------------------------------
// main program
// ------------------------------------------------------------------

int main(int argc, char *argv[]){

   if (argc == 1 || *argv[1] == '?'){
      printf("\ngauss␣v1.4,␣2010-12-03\n");
      printf("—␣Copyright␣(C)␣2009-2010␣Berndt␣E.␣␣Schwerdtfeger␣—␣\n\n");
      printf("␣␣Input:␣␣␣␣␣␣year-mm-dd␣[offset]␣\n\n");
      printf("␣␣Output:␣␣␣␣␣Weekday,␣date,␣Julian␣Calendar␣date,␣day␣of␣the␣year,\n");
      printf("␣␣␣␣␣␣␣␣␣␣␣␣␣␣week␣of␣the␣year,␣Julian␣day␣number,␣Unix␣day␣number\n\n");
      printf("␣␣Sample:␣gauss␣2010-01-00␣+120\n");
      printf("␣␣Fri␣2010-04-30,␣JC␣2010-04-17,␣D#␣120␣W#␣17␣J#␣2455317␣X#␣14729\n\n");

    return EXIT_SUCCESS;                // end the program
   }                                    // argc > 1 here

   int  w,m,d,jm,jd;
   long y,jy,j,x,n=0;
   char line[80];

   sscanf(argv[1],"%ld-%d-%d",&y,&m,&d);

   if (argc == 3)
      sscanf(argv[2],"%ld",&n);         // get the offset
   else if (argc > 3) {
      printf("Too␣many␣parameters␣!\n");
      return EXIT_FAILURE;              // end the program
   }

   while(1)
   {
      n += offset(y,m,d);               // correct the offset
      date(GREGOR, n, &y, &m, &d);      // correct y, m, d
      j = julian(y,m,d);                // set Julian day number
      x = j - 2440588;                  // set Unix day number
      jy = -4712;                       // set initial julian year
      date(JULIAN, j+1, &jy, &jm, &jd); // correct jy, jm, jd
      n = offset(y,m,d);                // set offset in this year
      w = weekday(y,m,d);               // set weekday (Gauss)

      printf("%s␣%ld-%02d-%02d,␣JC␣%ld-%02d-%02d,␣",wd[w],y,m,d,jy,jm,jd);
      printf("D#␣%03d␣W#␣%02d␣J#␣%ld␣X#␣%ld\n",n,(n-w+9)/7,j,x);
      fgets(line,sizeof line,stdin);    // read next
      if (line[0]=='\n')
         break;
      n = 0;
```

```
      sscanf(line,"%ld-%d-%d␣%ld",&y,&m,&d,&n);
   }                                    // end of while loop
   return EXIT_SUCCESS;                 // exit the program
};

// ------------------------------------------------------------------
// subroutines: date, offset, weekday, julian, leap, quot, mod
// ------------------------------------------------------------------
void date(int c, long n, long* y, int* m, int* d){

   int i;
   while (n > 365 + leap(c,*y)){        // if offset larger than # of
      n -= 365 + leap(c,*y);            // ... days in a year
      *y += 1;                          // ... find the correct year
   }

   while (n <= 0){                      // if offset is negative
      *y -= 1;                          // ... find the correct year
      n += 365 + leap(c,*y);            // ... and offset
   }

   i = leap(c,*y);                      // adjust for leap day
   *m = 12;
   while (n <= em[*m] + i){             // searching for the month
      *m -= 1;
      if (*m<3) i=0;
   }
   *d = n - em[*m] - i;                 // getting the day
}

// ------------------------------------------------------------------
long offset(long y, int m, int d){

   int x = d + em[m];                   // offset into this year
   if (m > 2)
      x += leap(GREGOR,y);              // adjust for leap day
   return x;
}

// ------------------------------------------------------------------
int weekday(long y, int m, int d){

   if (m < 3)
      y -= 1;

   long c = quot(y,100);
   int g = mod(y,100);
   int f = 5 * mod(c,4);
   int e = em[m];
   if (m > 2)
      e -= 1;
   return (-1 + d + e + f + g + g/4)%7; // Gauss' formula
```

(5/5) giving heat/recieving cold & vice versa -KEEP
UP NUTRITION/△ inspired/exercise /motion/blood flow [HOW TO
STAY EXCITED? unlikely:



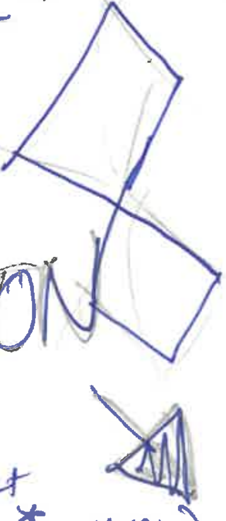
(law of cycles?) BUT
LOOK @ THIS CYCLE: (steps? upward onward)
STILL even if one doesnt descend or



one will STILL have to REST @ 0 ;)
PROOF? UP FOR PEER REVIEW
(even if you/it dont descend, & on X/y axis (that is
still "descending" on X axis (←) {want: ↑ → ↗;
(NOT: ↘ ↓ ↙ ← ↖) ///

(5/5) 10:17 not 11 10 47 laundry change current is 11:15
estimate ETA: 11:33 11:15 "if you {dont miss the heat} dont
listen to the heart - it can kill you (do all {ye's} come
from Shiva pigeon-Vulture?) /no, not all ;) /// laced joint
I'm not tryin to die! /// OH LOVE STORY VOMIT
SURVIVAL at one point I enjoyed such //
WHAT A SHIT STORY
"TO BE A BETTER CHRISTIAN >

IS MY RELIGION



(nuclear warfare would not make sense as opposed to micro foundations) Si
only a lil Retarded Today
THE ROD IS THE LAW 2

* not survival of strongest but of fittest/compatible* (other side)

Guest the Show is to BE the BEST Snitch / thief of SJB's life -(very few Good Samaritans, still enough to Survive) (Grateful)

As to (4): by symmetry $\lambda(-x) = \lambda(x)$, hence

$$\sum_{x=1+100c+g}^{0} \lambda(x) = \sum_{x=0}^{-1-100c-g} \lambda(x)$$

and we are in the situation of (3) with $-1-100c-g = 100(-1-c)+(100-1-g)$, $-1-c \geq 0$, $0 \leq 100-1-g < 100$,

$$\begin{aligned}\sum_{x=1+100c+g}^{0} \lambda(x) = 1 + \sum_{x=1}^{-1-100c-g} \lambda(x) =& \\ &= 1 + \left[\frac{-1-c}{4}\right] + 24(-c-1) + \left[\frac{100-1-g}{4}\right] = \\ &= 1 + \left[\frac{-1-c}{4}\right] - 24 \cdot c - 24 + 25 + \left[\frac{-1-g}{4}\right] = \\ &= -[c/4] - 24 \cdot c - [g/4]\end{aligned}$$

where we have used the relation $1 + \left[\frac{-1-n}{4}\right] = -\left[\frac{n}{4}\right]$, valid for any $n \in \mathbb{Z}$. ☐

2.3. **Shifting Dates.** The set T of valid 'dates' is a subset of

$$\mathbb{Z} \times \{1, \ldots, 12\} \times \{1, \ldots, 31\}$$

defined by

$$\begin{aligned} T := &\ \mathbb{Z} \times \{1,3,5,7,8,10,12\} \times \{1, \ldots, 31\} \\ &\cup \mathbb{Z} \times \{4,6,9,11\} \times \{1, \ldots, 30\} \\ &\cup \mathbb{Z} \times \{2\} \times \{1, \ldots, 28\} \\ &\cup \Lambda \times \{2\} \times \{29\} \end{aligned}$$

The last set in the union are the leap days (29th of February).

We have a finite 'fibration' map

$$\pi : T \longrightarrow \mathbb{Z}$$

given by the first projection $\pi(y, m, d) = y$. Normal fibers $\pi^{-1}(y)$, $y \notin \Lambda$, have 365 elements, whereas 'leap' fibers ($y \in \Lambda$) have 366, i.e. fiber cardinality is

$$|\pi^{-1}(y)| = 365 + \lambda(y) \tag{5}$$

We use the natural order of days: $t < t'$ if t' is later than t. The cardinality of the segment of the year of all days up to $t = (y, m, d)$ is

$$|\{t' \in \pi^{-1}(y) \mid t' \leq t\}| = d + e(m) + \lambda(y, m) \tag{6}$$

We have a natural *shift* operation of the additive group $\mathbb{Z}$ on T:

$$\begin{aligned} \sigma : \mathbb{Z} \times T &\longrightarrow T \\ (n, t) &\longmapsto n + t \end{aligned}$$

where $n + t$ is defined as t shifted by n days, earlier ($n < 0$) or later ($n > 0$) in time. We also write $\sigma_t(n) = \sigma(n, t) = n + t$, as well as $t' - t = n$ for $t' = n + t$.

Examples

$$\begin{aligned} 1 + (2000, 2, 28) &= (2000, 2, 29) \\ 1 + (2001, 2, 28) &= (2001, 3, 1) \\ -365 + (2001, 1, 1) &= (2000, 1, 2) \\ -152930 + (2001, 6, 30) &= (1582, 10, 15) \\ 1872 + (1947, 2, 4) &= (1952, 3, 21) \\ (y, m, d) &= d + e(m) + \lambda(y, m) + (y-1, 12, 31) \end{aligned}$$

VIETNAMIA
MAMIA
NOM NOM
NOMIA

This operation is simply transitive. Therefore, any $t \in T$ gives rise to an isomorphism

$$\begin{aligned} \sigma_t : \mathbb{Z} &\xrightarrow{\sim} T \\ n &\longmapsto n + t \end{aligned}$$

which is *linear*: $\sigma_t(n + m) = n + \sigma_t(m)$. Taking the inverse σ_t^{-1} and combining with $\mathbb{Z} \twoheadrightarrow \mathbb{F}_7$ gives us *Gauss* maps

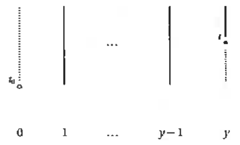
$$\omega_t : T \longrightarrow \mathbb{F}_7$$

which have the property $\omega_t(n + u) = n + \omega_t(u) = \omega_{-n+t}(u)$. So, there are only 7 of them, differing from each other by a constant $\in \mathbb{F}_7$.

Now I choose $t_0 = (0, 12, 31)$ as the arbitrary origin of our counting.

Let $t = (y, m, d) \in T$ be any date. To calculate $\omega(t) = \omega_{t_0}(t)$ we need to determine $n \in \mathbb{Z}$ with $n = t - t_0$. Then $\omega(t) = n \bmod 7 \in \mathbb{F}_7$.

To this end, we will have to go thru the fibers from t to t_0, from the fiber $\pi^{-1}(y)$ to the fiber $\pi^{-1}(1)$, and keep track of the number of days in passing.



(in this picture we assume $y > 0$).

Proposition 2.2.

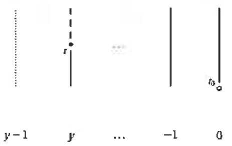
$$t - t_0 = d + e(m) + 365 \cdot (y-1) + [c/4] + 24 \cdot c + [g/4] \tag{7}$$

Proof. Let us first assume $y > 0$, then by (6) and (5)

$$\begin{aligned} t - t_0 &= d + e(m) + \lambda(y, m) + \sum_{x=1}^{y-1} |\pi^{-1}(x)| = \\ &= d + e(m) + \lambda(y, m) + \sum_{x=1}^{y-1} (365 + \lambda(x)) = \\ &= d + e(m) + 365 \cdot (y-1) + \begin{cases} \sum_{x=1}^{y} \lambda(x) & m \geq 3 \\ \sum_{x=1}^{y-1} \lambda(x) & m = 1,2 \end{cases} \\ &= d + e(m) + 365 \cdot (y-1) + \sum_{x=1}^{100c+g} \lambda(x) = \\ &= d + e(m) + 365 \cdot (y-1) + [c/4] + 24 \cdot c + [g/4] \end{aligned}$$

where we have substituted $y = 100 \cdot c + g$ resp. $y - 1 = 100 \cdot c + g$, and applied the Lemma in section 2.2, relation (3).

Let us now assume $y \leq 0$, we are in the situation of the following picture:



and counting similarly the fibers we get

$$\begin{aligned} t - t_0 &= d + e(m) + \lambda(y, m) - \sum_{x=y}^{0} |\pi^{-1}(x)| = \\ &= d + e(m) + \lambda(y, m) - \sum_{x=y}^{0} (365 + \lambda(x)) = \\ &= d + e(m) + 365 \cdot (y-1) - \sum_{x=1+100c+g}^{0} \lambda(x) = \\ &= d + e(m) + 365 \cdot (y-1) + [c/4] + 24 \cdot c + [g/4] \end{aligned}$$

where we have substituted $y = 100 \cdot c + g$ resp. $y - 1 = 100 \cdot c + g$, and applied the Lemma in section 2.2, relation (4). ☐

Corollary 2.3. *(Gauss formula)*

$$\omega(y, m, d) = d + e + f + g + [g/4] \tag{8}$$

where $f = 5 \cdot r$ *when* $c = q \cdot 4 + r, 0 \leq r \leq 3$ *and* $e = \begin{cases} e(m) & \textit{for } m = 1,2 \\ e(m) - 1 & \textit{for } m > 2 \end{cases}$

Proof. We have $\omega(t) = t - t_0 \mod 7$, hence by the proposition

$$\omega(t) = d + e(m) + y - 1 + [c/4] + 3 \cdot c + [g/4]$$

For $m = 1, 2$, as well as for $m > 2$

$$e(m) + y - 1 = e(m) - 1 + y = e + 100 \cdot c + g$$

therefore

$$\begin{aligned} \omega(t) &= d + e + 5 \cdot c + [c/4] + g + [g/4] = \\ &= d + e + 5 \cdot (q \cdot 4 + r) + q + g + [g/4] = \\ &= d + e + f + g + [g/4] \end{aligned}$$

☐

2.4. **The Julian day number.** The *Julian day number* is defined as the number of days passed since JC -4712-01-01, which is -4713-11-24 in the proleptic Gregorian Calendar, hence is given by $j(t) = t - t_1 = (t - t_0) - (t_1 - t_0)$, where $t_1 = (-4713, 11, 24)$. From equation (7) we evaluate

$$t_1 - t_0 = 24 + e(11) + 365 \cdot (-4713 - 1) + [-48/4] + 24 \cdot (-48) + [87/4] = -1721425$$

hence

$$j(t) = 1721060 + d + e(m) + 365 \cdot y + [c/4] + 24 \cdot c + [g/4] \tag{9}$$

REFERENCES

[1] Carl Friedrich Gauss, *Werke* (1863), available at `https://gdz.sub.uni-goettingen.de/id/PPN235957348`.

[2] Maurice Kraitchik, *Mathematical Recreations*, George Allen & Unwin Ltd., London, 1955.

APPENDIX A. THE FORMULA IN GAUSS' NACHLASS

GAUSS never published his formula for finding the day of the week. It appeared in 1927 in [1, XI, 1.] p. 206 with remarks by ALFRED LOEWY regarding its derivation.

A.1. Den Wochentag des 1. Januar eines Jahres zu finden.

Handschriftliche Eintragung in: Sammlung astronomischer Tafeln, unter Aufsicht der Kgl. Preußischen Akademie der Wissenschaften, I. Band, Berlin 1776. — 1798 von GAUSS erworben.

Bezeichnet man den kleinsten positiven Rest einer Grösse A nach dem Modulus m durch $R : A$ (mod m), so lassen sich alle Vorschriften des Gregorianischen Kalenders auf folgende geschmeidige Art darstellen:

1.

Wenn man die Tage vom 1ten Januar 1701 an zählt, d. i. diesen mit 1, den 2ten mit 2, den 31ten Dec. 1700 mit 0, den 30ten mit -1 etc. bezeichnet, so ist der 1te Januar in irgend einem Jahre A

$$\begin{aligned} = 1 + (A - 1701)365 + \frac{1}{4}\{(A - 1701) - R : (A - 1701) \bmod 4\}& \\ - \frac{1}{100}\{(A - 1701) - R : (A - 1701) \bmod 100\}& \\ + \frac{1}{400}\{(A - 1601) - R : (A - 1601) \bmod 400\}& \end{aligned}$$

2.

Die Wochentage Sonntag, Montag, etc. mit 0,1 etc. bezeichnet, ist der 1te Januar irgend eines Jahres, qua Wochentag,

$$\left.\begin{aligned} \equiv 6 + A + (2A + 5R : (A-1) \bmod 4)& \\ + (3A + 4R : (A-1) \bmod 100)& \\ + (A + 2 + 6R : (A-1) \bmod 400)& \end{aligned}\right\} \bmod 7.$$

Also von 1601 bis 2000

$$\equiv 6 + 6A + 5R : (A-1) \bmod 4 + 4R : (A-1) \bmod 100.$$

Allgemein

$$\equiv 1 + 5R : (A-1) \bmod 4 + 4R : (A-1) \bmod 100 + 6R : (A-1) \bmod 400.$$

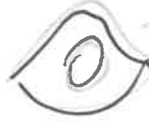
A.2. **Comparison.** The reader may look up the remarks by LOEWY in [1]. I contend that the formula in A.1 is identical to (8). Using earlier notation, let $y = A - 1 = 100c + g$ and $c = 4q + r$, then in GAUSS' notation $R : y \bmod m = y - [y/m]m$, hence

$$\begin{aligned} R : y \bmod 4 = R : g \bmod 4 &= g - 4[g/4] \\ R : y \bmod 100 &= g \\ R : y \bmod 400 &= 100r + g \end{aligned}$$

Calculating in $\mathbb{F}_7$ the weekday of January 1st of year A by A.1 is

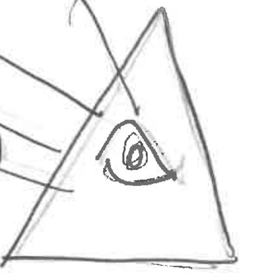
$$\begin{aligned} &= 1 + 5R : y \bmod 4 + 4R : y \bmod 100 + 6R : y \bmod 400 = \\ &= 1 + 5g - 20[g/4] + 4g + 12r + 6g = \\ &= 1 + g + [g/4] + 5r = \omega(A, 1, 1) \qquad \text{by formula (8).} \end{aligned}$$ ☐

(5/5) WHØSE DØING THE BE$T WATCHING!



I~~A~~M

ME ME ME



(Surveillance Game / Shit Show)

The Law Can be Quite Lengthy! TO STUDY

BUT Some Come First!

Survival & Biological Pre-Requisites (Sorry Spencer, we didnt study

Psuedo Community & addiction &/or Homelessness

Belonging

ENOUGH

5/5 Wormhole head* Squeez

WORMHOLE*

$2 could DO IT?

WE AINT TRYIN TO DIE — new fetty doctors on the block?

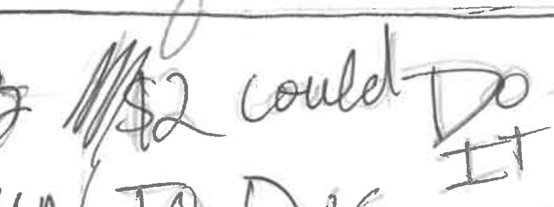
Drug edu & safety

STOP BILLIONARE -CONSCIOUSness IDENTITY THEFT → Transferring consciousness to little people!!

WRITERS VS [illegible] SCIENTISTS

MAKE knife blade/handle &/or "hour"/Time glasses

Castes to easily melt glass into



just need 1 caste for both

make castes out of clay AND for other materials

(wax & stuff)

AND recycled aluminum cans

what comes along &/ND 10¢ in WASHINGTON

whatcom & along

stacked along whatcom creeks

GAUSS' CALENDAR FORMULA FOR THE DAY OF THE WEEK

BERNDT E. SCHWERDTFEGER

In memory of my father Martin Walter (1910–1977)

Abstract. This paper describes a formula of C. F. Gauss for calculating the day of the week from the calendar date and gives a conceptual proof. A formula for the *Julian day number* is derived. My implementation of the formula in REXX and C is included.

Preface

Gauss' formula calculates the day of the week, *Monday, …, Sunday,* from the calendar date `year-month-day`. I learned about it in 1962 when reading the *Mathematical Recreations* [2] by Maurice Kraitchik (1882–1957). It is stated there without proof, together with an application to a perpetual calendar.

This article explains the formula and its mathematical background, including a proof and the concepts behind the algorithms implemented in the program `gauss.c`.

Gauss never published his formula; it appeared only in 1927 in his *Werke* [1, XI, 1.]. For the convenience of the reader I have included this notice in appendix A.1

Berlin, 11 May 2009

© 2001–2018 Berndt E. Schwerdtfeger

v1.4a, 14 November 2018

1. Gauss' formula explained

1.1. **The Gauss formula.** Given are d = day, m = month, y = year. We split the year into the century part and the two digit year inside the century. For $m \geq 3$ it is done for the year $y = 100 \cdot c + g$, i.e. $c = [y/100]$, $g = y - 100 \cdot c$, such that $0 \leq g \leq 99$. For $m = 1,2$ it is done for the year $y - 1 = 100 \cdot c + g$.

Following Kraitchik [2, The Calendar, p. 110–111] Gauss' formula is:

$$w \equiv d + e + f + g + [g/4] \pmod 7$$

where $w \equiv 1, \ldots$ corresponds to *Monday*, …, etc., and with the following constants depending on the month (e) resp. century (f):

Month	m	1	2	3	4	5	6	7	8	9	10	11	12
	e	0	3	2	5	0	3	5	1	4	6	2	4

Gregorian Calendar	Century	c mod 4	f
	1600, 2000, …	0	0
	1700, 2100, …	1	5
	1800, 2200, …	2	3
	1900, 2300, …	3	1

2010 *Mathematics Subject Classification.* Primary 68-04; Secondary 68N15.
Key words and phrases. Calendar formula, perpetual calendar, Julian day number.

For the *Julian calendar* the same formula applies with modified constants f according to the following table:

Julian Calendar	Century	c mod 7	f
	0000, 0700, 1400, …	0	5
	0100, 0800, 1500, …	1	4
	0200, 0900, 1600, …	2	3
	0300, 1000, 1700, …	3	2
	0400, 1100, 1800, …	4	1
	0500, 1200, 1900, …	5	0
	0600, 1300, 2000, …	6	6

Some *perpetual calendars* exploit this formula (or present tables that are equivalent to the formula). A short explanation (with some gaps) of this formula is given in the next section 1.2 and a complete proof in section 2.

1.2. **A short reasoning.** First, recall the Gregorian rule for *leap years:*

A normal year has 365 *days, with an extra day – the leap day 29.02. – added every four years, except for century years, when the century is not divisible by* 4.

In terms of our variables this reads

$$\begin{aligned} y = 100 \cdot c + g \quad & \text{is a leap year} :\Longleftrightarrow \\ & (g \neq 0 \text{ and } g \equiv 0 \bmod 4) \text{ or } (g = 0 \text{ and } c \equiv 0 \bmod 4) \end{aligned}$$

This rule implements a periodicity of 400 years that have exactly $3 \cdot 24 + 25 = 97$ leap years and $3 \cdot 76 + 75 = 303$ normal years. Otherwise put, 400 consecutive years in the Gregorian Calendar consists always of exactly $97 \cdot 366 + 303 \cdot 365 = 146097$ days, which results in an average length of a Gregorian year of $\frac{146097}{400} = 365.2425$ days (or, if you prefer, has 31,556,952 seconds).

A brief 'reasoning' on this formula (with some shortcuts) runs as follows:

The first term d, the progression with days, is obvious.

$365 \equiv 1 \pmod 7$, so the first and last day of any normal year fall on the same weekday. This explains the term g (from year to year progress). The other term $[g/4]$ accounts for the extra day every four years.

How much week days do we have to shift from one century to the next? $100 \equiv 2 \pmod 7$, plus the 24 leap years in the century give $24 \equiv 3 \pmod 7$, so a century adds $2 + 3 = 5$ shift days.

For the Julian Calendar we have 25 leap years in a century, thus a century shifts by $2 + 4 = 6$ days.

This explains the f values: $5 + 5 \equiv 3 \pmod 7$, $5 + 5 + 5 \equiv 1 \pmod 7$. Their dependency on $c \pmod 4$ and the gap ('leap') from $c \equiv 3$ to $c \equiv 0 \pmod 4$ is due to the leap century rule (in the Gregorian Calendar).

This leaves us to explain the 'magic' values for e. Let us look at the following table with the number $d(m)$ of days in a month, the sum $e(m)$ of days for the previous months (number of days from begin of the year for non leap years only) and their values mod 7:

m	1	2	3	4	5	6	7	8	9	10	11	12
$d(m)$	31	28	31	30	31	30	31	31	30	31	30	31
$e(m)$	0	31	59	90	120	151	181	212	243	273	304	334
(mod 7)	0	3	3	6	1	4	6	2	5	0	3	5

The 'magic' value is $e = e(m)$ for the months $m = 1,2$ and $e = e(m) - 1$ for the other months … (mod 7).

It is left to the reader as an exercise to render this reasoning into an irreproachable mathematical proof and to develop a more conceptual approach.

Instead, you might as well study the program in the appendix B.2: the function `offset` in `gauss.c` gives a different parametrization of dates in terms of *offset* and *year* ; the function `date` is inverse to `offset`.

Or, you might want to read the conceptual approach developed in section 2.

1.3. **Historical Perspective.** The actual internationally used calendar is derived from the Roman calendar as reformed by Julius Caesar in 46 BC, who introduced the leap day every four years (*Julian Calendar*). The average length of the Julian year is therefore 365.25 days.

Now, the *solar* (or *tropical*) year, which is founded on the revolution of the Earth around the Sun, defined by passing the *vernal equinox*, is measurably shorter: 365 days, 5 hours, 48 minutes and 45.51 seconds, or 31,556,925.51 s, or 365.242193402 days. This is about 11 minutes and 15 seconds shorter than the Julian average year.

In the course of the centuries this discrepancy added up, leading to the fact that the Julian year was behind schedule: it had inserted too many leap days. In the 16th century the deviation amounted to about 12 days. Ugo Boncompagni (Bologna 1502 – Roma 1585), better known as Pope Gregory XIII (1572), established in 1582 a calendar reform (*Gregorian Calendar*), which dropped 10 days: the day following (Julian) October 4 became (Gregorian) October 15, and introduced the leap year rules described above.

Note, however, the Gregorian year is not perfect for all times. It is (again) too long: but this time by only 26.49 seconds, but this will also add up over time. The imperfection accumulates to a whole day in about $1/(365.2425 - 365.242193402) = 3261$ years. Then, an additional leap day would have to be left out again … With the advent of leap *seconds* this will, of course, not happen: clocks nowadays tick much more accurate than heavenly bodies once did (they didn't – but we didn't know).

As the scale for numbering the years is changing occasionally in our History, we will have to make a choice. In this paper we will treat all Calendar dates as if the Gregorian Calendar is valid thru eternity (so called *proleptic* Gregorian Calendar). We will also make use of the astronomical numbering of years by integers $\ldots, -3, -2, -1, 0, 1, 2, \ldots$ (instead of the Historians' numbering of 4 BC, 3 BC, 2 BC, 1 BC, 1 AD, 2 AD). Ignoring the possibly finite number of years since Creation, we will use the rational integers for the years.

The program `gauss.c` ignores the historical truth as well and neglects the Julian rules in force from 46 BC up to 1582 (and beyond in several countries). Instead, it treats all input dates according to Gregorian rules, but it gives the date of the Julian Calendar in its output.

2. Towards a conceptual approach

2.1. **Heuristical remarks.** In principle the days from now to the past or to the future look like the set $\mathbf{Z}$ of rational integers. But there is no natural *day zero* and one has to make an arbitrary choice. We will see one way to do this numbering in section 2.4.

The set T of dates is another funny renumbering of $\mathbf{Z}$ by a triple $t = (y, m, d)$ of year, month, day (in some countries noted as `d.m.y`, in others `d/m/y` or even `m/d/y`; the ISO 8601 format prescribes `y-m-d` and this is also used in my program `gauss.c`.

The searched for map from T to the set of weekdays {Monday, …, Sunday} should be as simple as the canonical map $\mathbf{Z} \to \mathbf{F}_7$, the integers (mod 7). We will construct a candidate, to be called *Gauss maps*

$$\omega : T \longrightarrow \mathbf{F}_7$$

in a fairly natural way and give an explicit formula in terms of the coordinates y, m, d — it turns out to be the *Formula of Gauss* (8) in Corollary 2.3.

2.2. **The leap function.** For a year $y \in \mathbf{Z}$ we set the *century* $c = [y/100]$ and the remainder $g = y - 100 \cdot c$, such that $0 \leq g \leq 99$. The (Gregorian) leap function on the set of years is a boolean map

$$\lambda : \mathbf{Z} \longrightarrow \{0, 1\}$$

given by

$$\lambda(y) = \begin{cases} 1 & \text{if} \quad (g \neq 0 \wedge g \equiv 0 \bmod 4) \vee (g = 0 \wedge c \equiv 0 \bmod 4), \\ 0 & \text{otherwise} \end{cases}$$

The subset of leap years will be denoted by

$$\Lambda = \{y \in \mathbf{Z} \mid \lambda(y) = 1\}$$

We have

$$\Lambda = 400\mathbf{Z} \sqcup \bigsqcup_{k \not\equiv 0 \bmod 25} 4k + 100\mathbf{Z}$$

We also define for a month m and any year y

$$\lambda(y, m) = \begin{cases} \lambda(y) & \text{if } m > 2 \\ 0 & \text{for } m = 1, 2 \end{cases}$$

As we need to keep track of counting leap years thru the centuries, let us note some properties of λ, more or less obvious from the definitions.

Lemma 2.1. *Let* $0 \leq g < 100$. *We have*

$$\sum_{l=1}^{c} \lambda(100 \cdot l) = [c/4] \qquad c \geq 0 \tag{1}$$

$$\sum_{k=1}^{g} \lambda(100 \cdot c + k) = [g/4] \tag{2}$$

$$\sum_{x=1}^{100c+g} \lambda(x) = [c/4] + 24 \cdot c + [g/4] \qquad c \geq 0 \tag{3}$$

$$-\sum_{x=1+100c+g}^{0} \lambda(x) = [c/4] + 24 \cdot c + [g/4] \qquad c < 0 \tag{4}$$

Proof. The relations (1), (2) are obvious by the leap year rule. As to (3):

$$\begin{aligned} \sum_{y=0}^{100c+g} \lambda(y) &= \sum_{l=0}^{c-1} \sum_{k=0}^{99} \lambda(100 \cdot l + k) + \sum_{k=0}^{g} \lambda(100 \cdot c + k) \\ &= \sum_{l=0}^{c} \lambda(100 \cdot l) + \sum_{l=0}^{c-1} \sum_{k=1}^{99} \lambda(100 \cdot l + k) + \sum_{k=1}^{g} \lambda(100 \cdot c + k) \\ &= [c/4] + 1 + \sum_{l=0}^{c-1} 24 + [g/4] \\ &= [c/4] + 1 + 24 \cdot c + [g/4] \qquad \text{and } \lambda(0) = 1. \end{aligned}$$

spilled in pocket

(5/5) ketchup packets, uncovered shits, Scientific vials

tobacco dip container biting off/out cotton

and Culture Dishes

filter cigarette Sacred/consensual circumsisions //

I'M NOT GONNA GIVE CIRCUMSISONS, ID rather baptize

dead People (skeletons, well cleansed) /// ☒ Read about Jesus

words on cir'ss, not Paul

/// ☐ Research Crank Movie

// ☐ DECREASE CHANCE OF SINNING multi-use

50 cal (6, 9, 12 rounds cartridge sizes for 3x Burst

Shot) Babyfug VHS LOGO'd in secret compartment (micro VHS)

☐ LESS RECOIL, LIGHTER WEIGHT, smaller size

☒ 50 cal heat seeker

☐ open sourced Gun, multiple molds/caster lighter weight/modular (glass, clay, etc, other metals/materials

△ ☐ Demand Universal Gift/Trophee

△ "Thanks for Surviving"... "heres

△ all your Records, WE Stole from

△ you"... (surveillance State apologies,

△ Post 9/11) (data storage eliminates

△ cost of paper/needless expense)

/// The Hierarchy/priorities of The Back pack

(see small notebook SJBCC2605 /// chanting

Beliefs ▽ of Knowledge △ (feedback loops)

Authenticity evolution

T

kt24601 • 10y ago
The mod makes it difficult
⇧ 1 ⇩

r/UndeadUnluck • 2y ago
SPOILER
Un-Order Game Day 1 (Sakamoto Days Spoilers)
47 upvotes · 7 comments

r/televisionsuggestions • 2mo ago
Any foreign period drama produced in France, Russia, Germany or other european country?
14 upvotes · 37 comments

r/PLC • 6y ago
Everything in the automation industry

796 upvotes · 52 comments

PROMOTED ...

Answers and care all in one chat.
Learn more
amazon

r/cpp • 6mo ago
A Faster Algorithm for Date Conversion
75 upvotes · 29 comments

VIEW POST IN
Português (Brasil)
Русский
简体中文
日本語
See more ⌄

Reddit
reReddit: Top posts of June 14, 2016

Reddit
reReddit: Top posts of June 2016

Reddit
reReddit: Top posts of 2016

2

(5/6) ☿ found that stringy ①

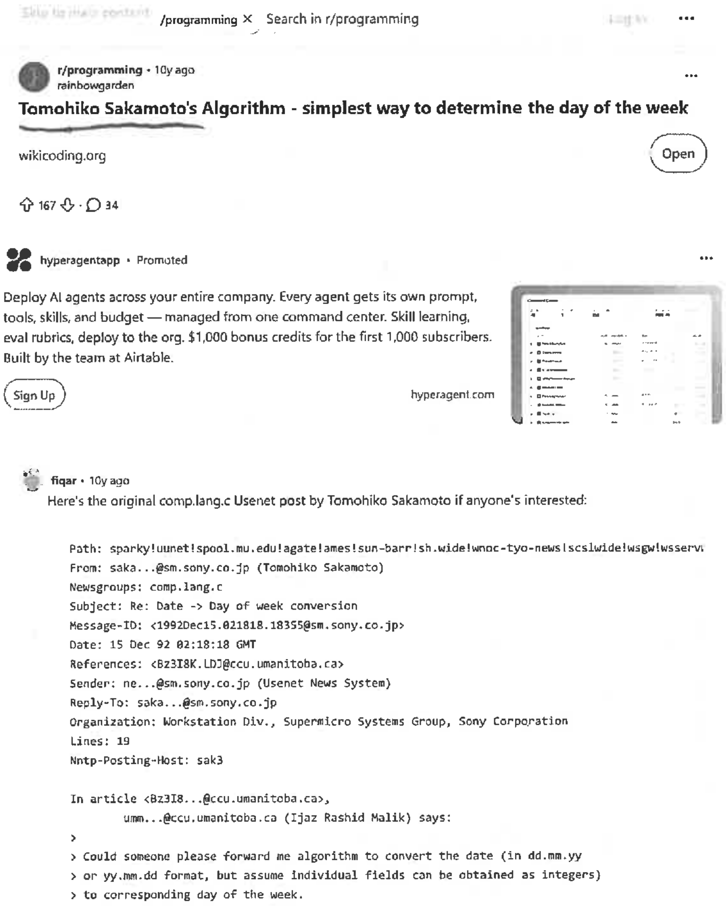
lettuce "vulva" seaweed
only ate about 2 strands
&/or 2 feet worth to build
biome to this dirty Watershed
after seeing geese eat it yester
day. /// Opensource / De-mono-
polize Police force BECAUSE
A) monopolies are bad B) ~~women~~ witches
seduce them to do their dirty
work c) everyone should ~~to~~
the skill to assault, bond/bind,
and entrap in cages (NOT
MEDICINE / FIRST AID ✚)
d) /// The Biggest & Most valuable
Lesson I learned from the
U.S.A. is if you can't handle
the Government fucking your
Babies you should kill yourselves //

DONT WORRY THE USA CAN (
AND WILL FRAME ANYONE
AND EVERYONE AS A BABYFUQ?
then you will see you have no
real friends [Loyal / Love] just
like Epstein & Spencer Joseph
Barrett ✗ cancel statement – you
$ just need money to buy friend
SO JUST POOR PPL like Spencer
are fucked ... ☐ more Research
on alleged Friendships ///
"When I was a child, I used to
give a fuck about the Govern-
ment fucking my babies, but
when I became a man I
realized I have no Power" – Sa
/ (fuck paul – Jesus) ////
☐ How much with all my semen
jizz over lifetime (racoon) tried to
stop this writing or helped? ~5–10 min aft
orgasm toe tou

☐ Phantasmagoria [YT all cap] ③
/// 11:45 pm (5/6) 5 min to Re-
SPECT wheelchair Camper in
the outhouse @ Bus Stop –
(will give 10 minutes), met
Lil honor actress Ella & her
Guitar friend, she seemed LIKES ME
friendly, PROBABLY USED AS BAIT no eyebrows, slight
Kenna face, (#KennaCatz)
she knew Harold, Adam,
Vander & Pandey, claimed
Rainbow connections, told her
of my hatred towards them
trafficking my daughter
# gave her GOFUNDME
image signed of me
Beauty – after she gave me
a Quit

– Like weather Told me [illegible] boat
[illegible] later

/// (5/6) well, in the time I (
shut up, (now (5/7) 12:03) we
became friends or I spoke on
my baby chickens n the one
sole survivor I still think abo
Lil Peep, she said she be out
in 5 min, but she got a wheel-
chair & probably gets paid to
camp out in there... // Great
† Reading today and Woe to them
† that travel in the cold/winter/sno
† just opened to Mark 13:16 & then
† gained context by starting from chapt
13, the Reading the other day
§ where I disclosed the first &
oldest written laws/commands
§ Gen 1:28 be fruitful & Multiply
§ 1:28 Replenish The Earth
§ /2:15 put into the Garden of
Eden to dress it & to keep it S



r/programming • 10y ago
rainbowgarden

Tomohiko Sakamoto's Algorithm - simplest way to determine the day of the week

wikicoding.org

⇧ 167 ⇩ · 34

hyperagentapp • Promoted

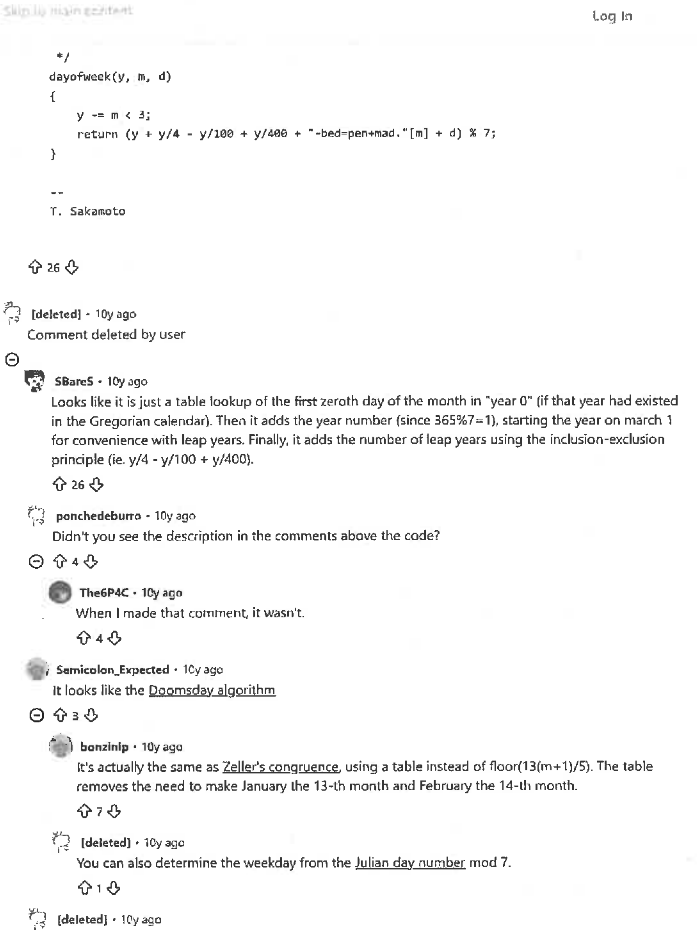
Deploy AI agents across your entire company. Every agent gets its own prompt, tools, skills, and budget — managed from one command center. Skill learning, eval rubrics, deploy to the org. $1,000 bonus credits for the first 1,000 subscribers. Built by the team at Airtable.

Sign Up

hyperagent.com

fiqar • 10y ago
Here's the original comp.lang.c Usenet post by Tomohiko Sakamoto if anyone's interested:

```
Path: sparky!uunet!spool.mu.edu!agate!ames!sun-barr!sh.wide!wnoc-tyo-news!scslwide!wsgw!wsserv
From: saka...@sm.sony.co.jp (Tomohiko Sakamoto)
Newsgroups: comp.lang.c
Subject: Re: Date -> Day of week conversion
Message-ID: <1992Dec15.021818.18355@sm.sony.co.jp>
Date: 15 Dec 92 02:18:18 GMT
References: <Bz3I8K.LDJ@ccu.umanitoba.ca>
Sender: ne...@sm.sony.co.jp (Usenet News System)
Reply-To: saka...@sm.sony.co.jp
Organization: Workstation Div., Supermicro Systems Group, Sony Corporation
Lines: 19
Nntp-Posting-Host: sak3

In article <Bz3I8...@ccu.umanitoba.ca>,
        umm...@ccu.umanitoba.ca (Ijaz Rashid Malik) says:
>
> Could someone please forward me algorithm to convert the date (in dd.mm.yy
> or yy.mm.dd format, but assume individual fields can be obtained as integers)
> to corresponding day of the week.
```



```
 */
dayofweek(y, m, d)
{
    y -= m < 3;
    return (y + y/4 - y/100 + y/400 + "-bed=pen+mad."[m] + d) % 7;
}

--
T. Sakamoto
```

⇧ 26 ⇩

[deleted] • 10y ago
Comment deleted by user

⊖

SBareS • 10y ago
Looks like it is just a table lookup of the ~~first~~ zeroth day of the month in "year 0" (if that year had existed in the Gregorian calendar). Then it adds the year number (since 365%7=1), starting the year on march 1 for convenience with leap years. Finally, it adds the number of leap years using the inclusion-exclusion principle (ie. y/4 - y/100 + y/400).

⇧ 26 ⇩

ponchedeburro • 10y ago
Didn't you see the description in the comments above the code?

⊖ ⇧ 4 ⇩

The6P4C • 10y ago
When I made that comment, it wasn't.

⇧ 4 ⇩

Semicolon_Expected • 10y ago
It looks like the Doomsday algorithm

⊖ ⇧ 3 ⇩

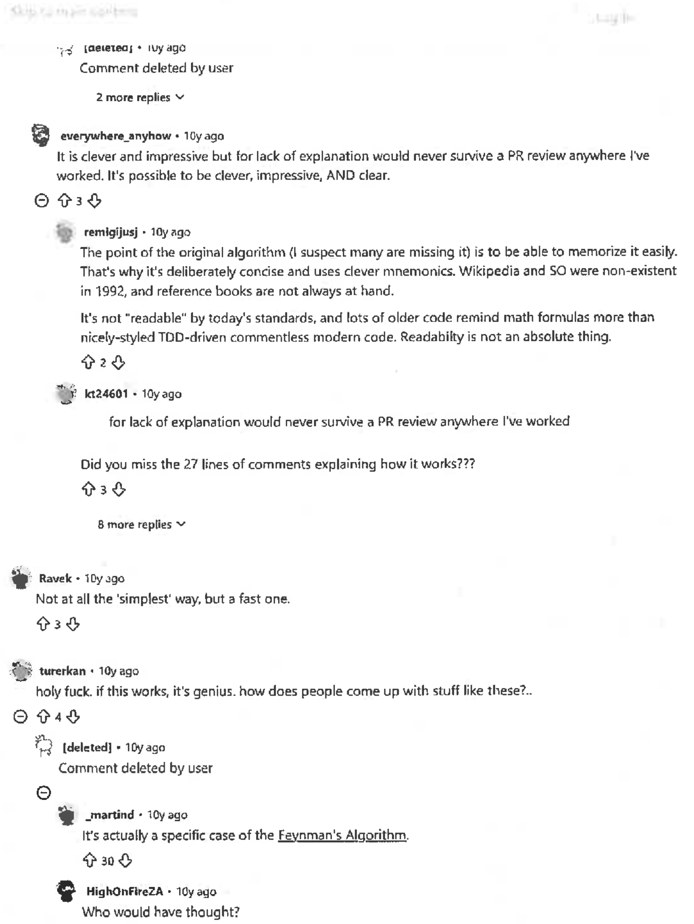
bonzinip • 10y ago
It's actually the same as Zeller's congruence, using a table instead of floor(13(m+1)/5). The table removes the need to make January the 13-th month and February the 14-th month.

⇧ 7 ⇩

[deleted] • 10y ago
You can also determine the weekday from the Julian day number mod 7.

⇧ 1 ⇩

[deleted] • 10y ago



[deleted] • 10y ago
Comment deleted by user

2 more replies ⌄

everywhere_anyhow • 10y ago
It is clever and impressive but for lack of explanation would never survive a PR review anywhere I've worked. It's possible to be clever, impressive, AND clear.

⊖ ⇧ 3 ⇩

remigijusj • 10y ago
The point of the original algorithm (I suspect many are missing it) is to be able to memorize it easily. That's why it's deliberately concise and uses clever mnemonics. Wikipedia and SO were non-existent in 1992, and reference books are not always at hand.

It's not "readable" by today's standards, and lots of older code remind math formulas more than nicely-styled TDD-driven commentless modern code. Readabilty is not an absolute thing.

⇧ 2 ⇩

kt24601 • 10y ago

> for lack of explanation would never survive a PR review anywhere I've worked

Did you miss the 27 lines of comments explaining how it works???

⇧ 3 ⇩

8 more replies ⌄

Ravek • 10y ago
Not at all the 'simplest' way, but a fast one.

⇧ 3 ⇩

turerkan • 10y ago
holy fuck. if this works, it's genius. how does people come up with stuff like these?..

⊖ ⇧ 4 ⇩

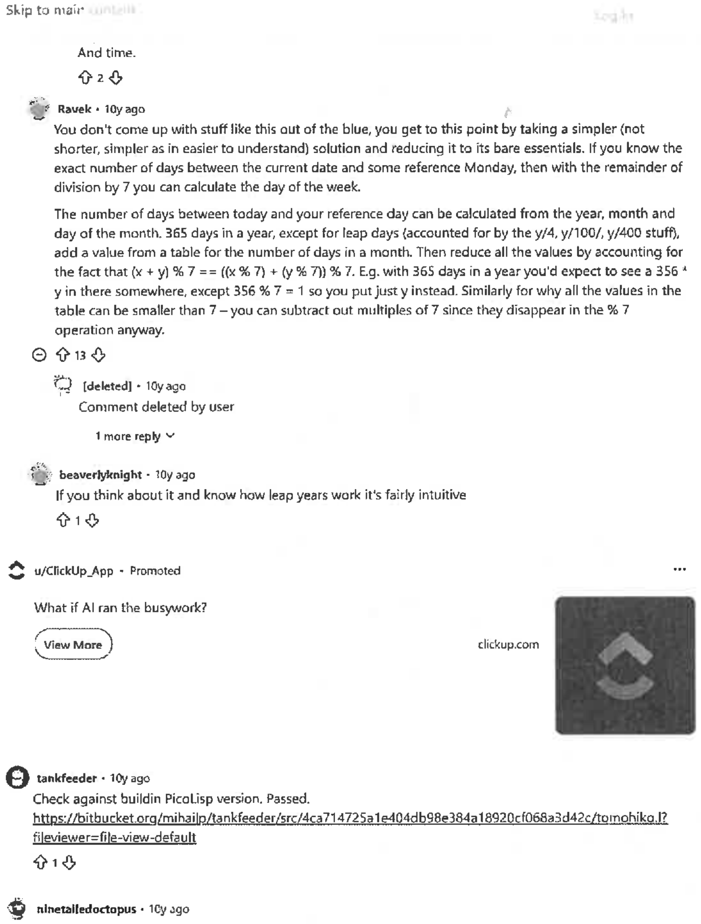
[deleted] • 10y ago
Comment deleted by user

⊖

_martind • 10y ago
It's actually a specific case of the Feynman's Algorithm.

⇧ 30 ⇩

HighOnFireZA • 10y ago
Who would have thought?



And time.

⇧ 2 ⇩

Ravek • 10y ago
You don't come up with stuff like this out of the blue, you get to this point by taking a simpler (not shorter, simpler as in easier to understand) solution and reducing it to its bare essentials. If you know the exact number of days between the current date and some reference Monday, then with the remainder of division by 7 you can calculate the day of the week.

The number of days between today and your reference day can be calculated from the year, month and day of the month. 365 days in a year, except for leap days (accounted for by the y/4, y/100/, y/400 stuff), add a value from a table for the number of days in a month. Then reduce all the values by accounting for the fact that (x + y) % 7 == ((x % 7) + (y % 7)) % 7. E.g. with 365 days in a year you'd expect to see a 356 * y in there somewhere, except 356 % 7 = 1 so you put just y instead. Similarly for why all the values in the table can be smaller than 7 – you can subtract out multiples of 7 since they disappear in the % 7 operation anyway.

⊖ ⇧ 13 ⇩

[deleted] • 10y ago
Comment deleted by user

1 more reply ⌄

beaverlyknight • 10y ago
If you think about it and know how leap years work it's fairly intuitive

⇧ 1 ⇩

u/ClickUp_App • Promoted

What if AI ran the busywork?

View More

clickup.com

tankfeeder • 10y ago
Check against buildin PicoLisp version. Passed.
https://bitbucket.org/mihailp/tankfeeder/src/4ca714725a1e404db98e384a18920cf068a3d42c/tomohiko.l?fileviewer=file-view-default

⇧ 1 ⇩

ninetailedoctopus • 10y ago

(8/5) 07 One of the Oldest Written/Record/command-ments

TEND the GARDEN

5.01

(5/7) 12:07 4// Really Sad to see this duck move so fast - I couldnt tell if it (female mallard) was upset/in-pain or/& fed drugs never Saw a duck move like that, and worried for my baby Geese (9:25am)///

The full 400-year cycle of doomsdays is given in the adjacent table. The centuries are for the Gregorian and proleptic Gregorian calendar, unless marked with a J for Julian. The Gregorian leap years are highlighted.

Julian centuries	-1600J	-1500J	-1400J	-1300J	-1200J	-1100J	-1000J
	−900J	−800J	−700J	−600J	−500J	−400J	−300J
	−200J	−100J	0J	100J	200J	300J	400J
	500J	600J	700J	800J	900J	1000J	1100J
	1200J	1300J	1400J	1500J	1600J	1700J	1800J
	1900J	2000J	2100J	2200J	2300J	2400J	2500J
	2600J	2700J	2800J	2900J	3000J	3100J	3200J
	3300J	3400J	3500J	3600J	3700J	3800J	3900J

Negative years use astronomical year numbering. Year 25BC is −24, shown in the column of −100J (proleptic Julian) or −100 (proleptic Gregorian), at the row 76.

Frequency of Gregorian doomsday in the 400-year cycle per weekday and year type

	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Total
Non-leap years	43	43	43	43	44	43	44	303
Leap years	13	15	13	15	13	14	14	97
Total	56	58	56	58	57	57	58	400

A leap year with Monday as doomsday means that Sunday is one of 97 days skipped in the 400-year sequence. Thus the total number of years with Sunday as doomsday is 71 minus the number of leap years with Monday as doomsday, etc. Since Monday as doomsday is skipped across February 29, 2000, and the pattern of leap days is symmetric about that leap day, the frequencies of doomsdays per weekday (adding common and leap years) are symmetric about Monday. The frequencies of doomsdays of leap years per weekday are symmetric about the doomsday of 2000, Tuesday.

The frequency of a particular date being on a particular weekday can easily be derived from the above (for a date from January 1 – February 28, relate it to the doomsday of the previous year).

For example, February 28 is one day after doomsday of the previous year, so it is 58 times each on Tuesday, Thursday and Sunday, etc. February 29 is doomsday of a leap year, so it is 15 times each on Monday and Wednesday, etc.

28-year cycle

Regarding the frequency of doomsdays in a Julian 28-year cycle, there are 1 leap year and 3 common years for every weekday, the latter 6, 17 and 23 years after the former (so with intervals of 6, 11, 6, and 5 years; not evenly distributed because after 12 years the day is skipped in the sequence of doomsdays). The same cycle applies for any given date from March 1 falling on a particular weekday.

For any given date up to February 28 falling on a particular weekday, the 3 common years are 5, 11, and 22 years after the leap year, so with intervals of 5, 6, 11, and 6 years. Thus the cycle is the same, but with the 5-year interval after instead of before the leap year.

Thus, for any date except February 29, the intervals between common years falling on a particular weekday are 6, 11, 11. See e.g. at the bottom of the page Common year starting on Monday the years in the range 1906–2091.

For February 29 falling on a particular weekday, there is just one in every 28 years, and it is of course a leap year.

Julian calendar

The Gregorian calendar is currently accurately lining up with astronomical events such as solstices. In 1582 this modification of the Julian calendar was first instituted. To correct for calendar drift, 10 days were skipped, so doomsday moved back 10 days (i.e. 3 weekdays): Thursday, October 4 (Julian, doomsday is Wednesday) was followed by Friday, October 15 (Gregorian, doomsday is Sunday). The table includes Julian calendar years, but the algorithm is for the Gregorian and proleptic Gregorian calendar only.

Note that the Gregorian calendar was not adopted simultaneously in all countries, so for many centuries, different regions used different dates for the same day.

Full examples

Example 1 (1985)

Suppose we want to know the day of the week of September 18, 1985. We begin with the century's anchor day, Wednesday. To this, add a, b, and c above:

- a is the floor of $\frac{85}{12}$, which is 7.
- b is 85 mod 12, which is 1.
- c is the floor of $\frac{b}{4}$, which is 0.

This yields $a + b + c = 8$. Counting 8 days from Wednesday, we reach Thursday, which is the doomsday in 1985. (Using numbers: In modulo 7 arithmetic, 8 is congruent to 1. Because the century's anchor day is Wednesday (index 3), and 3 + 1 = 4, doomsday in 1985 was Thursday (index 4).) We now compare September 18 to a nearby doomsday, September 5. We see that the 18th is 13 past a doomsday, i.e. one day less than two weeks. Hence, the 18th was a Wednesday (the day preceding Thursday). (Using numbers: In modulo 7 arithmetic, 13 is congruent to 6 or, more succinctly, −1. Thus, we take one away from the doomsday, Thursday, to find that September 18, 1985, was a Wednesday.)

Example 2 (other centuries)

Suppose that we want to find the day of week that the American Civil War broke out at Fort Sumter, which was April 12, 1861. The anchor day for the century was 94 days after Tuesday, or, in other words, Friday (calculated as $18 \times 5 + \lfloor \frac{18}{4} \rfloor$; or just look at the chart, above, which lists the century's anchor days). The digits 61 gave a displacement of six days so doomsday was Thursday. Therefore, April 4 was Thursday so April 12, eight days later, was a Friday.

See also

- Ordinal date
- Computus – Gauss algorithm for Easter date calculation
- Zeller's congruence – An algorithm (1882) to calculate the day of the week for any Julian or Gregorian calendar date.
- Mental calculation

References

1. John Horton Conway, "Tomorrow is the Day After Doomsday" (https://web.archive.org/web/20240907031643/https://www.archim.org.uk/eureka/archive/Eureka-36.pdf) (PDF). Eureka. October 1973. pp. 28–32.
2. Richard Guy, John Horton Conway, Elwyn Berlekamp : "Winning Ways: For Your Mathematical Plays, Volume. 2: Games in Particular", pages 795–797, Academic Press, London, 1982, ISBN 0-12-091102-7.
3. Lewis Carroll, "To Find the Day of the Week for Any Given Date", *Nature*, March 31, 1887. doi:10.1038/035517a0 (https://doi.org/10.1038%2F035517a0)
4. Martin Gardner, *The Universe in a Handkerchief: Lewis Carroll's Mathematical Recreations, Games, Puzzles, and Word Plays*, pages 24–26, Springer-Verlag, 1996.
5. "What Day is Doomsday" (https://ww2.amstat.org/mam/2014/calendar/doomsday.html). Mathematics Awareness Month. April 2014.
6. Alpert, Mark (April 1, 1999). "Not Just Fun and Games" (https://www.scientificamerican.com/article/not-just-fun-and-games/). *Scientific American*. Retrieved April 18, 2024.
7. The table is filled in horizontally, skipping one column for each leap year. This table cycles every 28 years, except in the Gregorian calendar on years that are a multiple of 100 (such as 1800, 1900, and 2100 which are not leap years) that are not also a multiple of 400 (like 2000 which is still a leap year). The full cycle is 28 years (1,461 weeks) in the Julian calendar and 400 years (20,871 weeks) in the Gregorian calendar.
8. Torrence, Bruce; Torrence, Eve. "John H. Conway – Doomsday, part 1" (https://www.youtube.com/watch?v=T_nQG-Bzxsg). *YouTube*. Mathematical Association of America. Archived (https://ghostarchive.org/varchive/youtube/20211221/T_nQG-Bzxsg) from the original on December 21, 2021. Retrieved April 14, 2020.
9. "The Doomsday Algorithm – Numberphile" (https://www.youtube.com/watch?v=z2x3SSBVGJU). *YouTube*. Retrieved July 9, 2023.
10. Limeback, Rudy (January 3, 2017). "Doomsday Algorithm" (http://rudy.ca/doomsday.html). Retrieved May 27, 2017.
11. "On what day of the week is Christmas? Use the Doomsday Rule" (https://www.irishtimes.com/news/science/on-what-day-of-the-week-is-christmas-use-the-doomsday-rule-1.4427431). *The Irish Times*. Retrieved July 20, 2022.
12. John Horton Conway, "Tomorrow is the Day After Doomsday" (https://www.archim.org.uk/eureka/archive/Eureka-36.pdf) (PDF). Eureka. October 1973. pp. 29–30. "Each ordinary year has its Doomsday 1 day later than the previous year, and each leap year 2 days later. It follows that within any given century a dozen years advances Doomsday by 12 + 3 = 15 days = 1 day. ('A dozen years is but a day.') So we add to the Doomsday for the century year the number of dozens of years thereafter, the remainder, and the number of fours in the remainder."
13. Chamberlain Fong, Michael K. Walters: "Methods for Accelerating Conway's Doomsday Algorithm (part 2)" (https://arxiv.org/abs/1010.0765), 7th International Congress on Industrial and Applied Mathematics (2011).
14. Nakai, Hirofumi (June 1, 2023). "A Simple Formula for Doomsday" (https://link.springer.com/article/10.1007/s00283-022-10229-3). *The Mathematical Intelligencer*. **45** (2): 131–132. doi:10.1007/s00283-022-10229-3 (https://doi.org/10.1007%2Fs00283-022-10229-3). ISSN 1866-7414 (https://search.worldcat.org/issn/1866-7414).
15. thatsmaths (June 22, 2023). "A Simple Formula for the Weekday" (https://thatsmaths.com/2023/06/22/a-simple-formula-for-the-weekday/). *ThatsMaths*. Retrieved March 21, 2025.

External links

- Encyclopedia of Weekday Calculation by Hans-Christian Solka, 2010 (https://web.archive.org/web/20061018103244/http://www.cs.wustl.edu/~ksl2/mathpoem.html)
- World records for mentally calculating the day of the week in the Gregorian Calendar (http://www.recordholders.org/en/records/dates.html)
- National records for finding Calendar Dates (http://www.recordholders.org/en/list/mental-calculation-rankings.html)
- World Ranking of Memoriad Mental Calendar Dates (http://www.memoriad.com/index.asp?s=kategoriler&b=kategori-detay&kategoriid=dcacea11f97125360e50694fd11c2ae4&lang=EN) (all competitions combined)
- Doomsday Algorithm (http://rudy.ca/doomsday.html)
- Finding the Day of the Week (https://web.archive.org/web/20070115153606/http://www.theworldofstuff.com/other/day.html)
- Poem explaining the Doomsday rule (https://web.archive.org/web/20061018103244/http://www.cs.wustl.edu/~ksl2/mathpoem.html) at the Wayback Machine (archived October 18, 2006)
- Doomsday Calculator (https://www.timeanddate.com/date/doomsday-calculator.html) at timeanddate

Retrieved from "https://en.wikipedia.org/w/index.php?title=Doomsday_rule&oldid=1347142408"